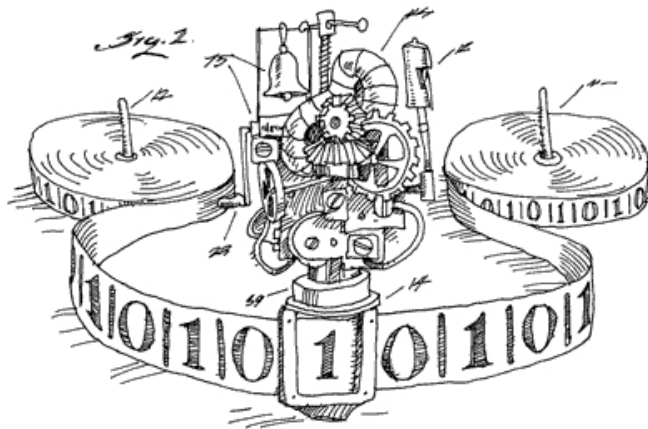


6. Turing Machines



Contents

- Turing Machines
- Language recognition
- Functions and Output
- Programming...

Turing Machines

- Two classes of machines, FSA and PDA, have so far been seen, along with their corresponding classes of language: regular and context-free.
- It has been observed that some languages, are not recognisable even by non-deterministic PDAs.
- The next class of machine, Turing Machines, is more powerful still.

Turing Machines

- In fact, it is the most powerful of all computational machines – if a Turing Machine cannot compute it, it is not computable!
- This **model of computation** was developed by ***Alan Mathison Turing*** in the 1930s !

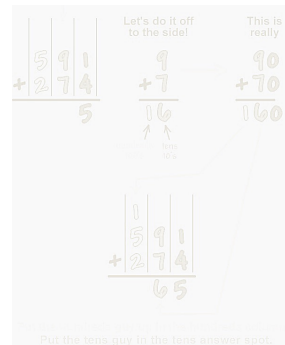


Turing, A. M. (1936).

“On Computable Numbers, with an Application to the Entscheidungsproblem”

Computation (..take a step back)

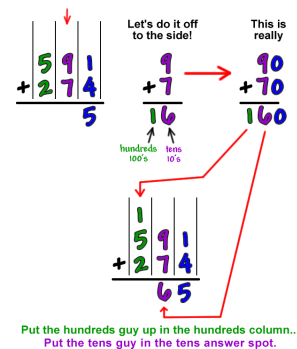
- What is computation ? What is an algorithm ?
- For example, how do we (humans) actually compute/calculate the sum of two numbers ?



- Read the numbers (digit per digit)
- Follow precise, fixed rules, i.e., an algorithm
- Sometimes, use a scratch pad for intermediate calculations
- Write the sum – final result to the scratch pad (and stop)

Computation (..take a step back)

- What is computation ? What is an algorithm ?
- For example, how do we (humans) actually compute/calculate the sum of two numbers ?



- Read the numbers (digit per digit)
- Follow precise, fixed rules, i.e., an algorithm
- Sometimes, use a scratch pad for intermediate calculations
- Write the sum – final result to the scratch pad (and stop)

Turing Machines: sketch

- A Turing Machine consists similar components to the other machines studied so far, that is, states, a tape and ‘some storage’.
- However, for this class of machine, the tape is the storage device – the tape can be both read-from and write-to.
- Furthermore, the current tape head (position) does not simply move from left to right (one direction), but can move left or right (both directions).

Turing Machines: sketch

- Another feature of Turing Machines is the way that they terminate computation.
- FSA and PDA carry out computation until they either crash (there is no applicable transition) or all of the input is consumed: Observation of the state (or the stack) indicates acceptance or otherwise.
- Turing Machines continue computation until they reach special states designated 'halt', at which point computation stops.

Turing Machines: sketch

- Another feature of Turing Machines is the way that they terminate computation.
- FSA and PDA carry out computation until they either crash (there is no applicable transition) or all of the input is consumed: Observation of the state (or the stack) indicates acceptance or otherwise.
- Turing Machines continue computation until they reach special states designated 'halt', at which point computation stops.

Definition

- A **Turing Machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$,
where $(H = \{q_{\text{accept}}, q_{\text{reject}}\})$:
 - Q is a set of states
 - Σ is the input alphabet, not containing $_$ ($_$ is the *blank symbol*)
 - Γ is the tape alphabet where $_ \in \Gamma$ and $\Sigma \subseteq \Gamma$.
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state (and $q_{\text{accept}} \neq q_{\text{reject}}$)
- $\delta: (Q - H) \times \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function

Definition

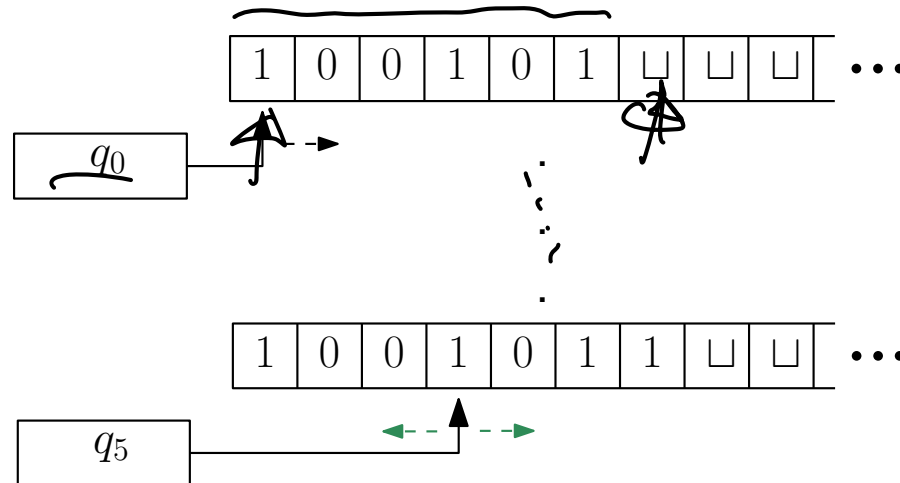
- **A Turing Machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $H = \{q_{\text{accept}}, q_{\text{reject}}\}$:
 - Q is a set of states
 - Σ is the input alphabet, not containing ' $_$ ' ($_$ is the *blank symbol*)
 - Γ is the tape alphabet where $_ \in \Gamma$ and $\Sigma \subseteq \Gamma$.
 - $q_0 \in Q$ is the start state
 - $q_{\text{accept}} \in Q$ is the accept state
 - $q_{\text{reject}} \in Q$ is the reject state (and $q_{\text{accept}} \neq q_{\text{reject}}$)
 - $\delta: (Q - H) \times \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function

Definition

- A **Turing Machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $H = \{q_{\text{accept}}, q_{\text{reject}}\}$:
 - Q is a set of states
 - Σ is the input alphabet, not containing ' $_$ ' ($_$ is the *blank symbol*)
 - Γ is the tape alphabet where $_ \in \Gamma$ and $\Sigma \subseteq \Gamma$.
 - $q_0 \in Q$ is the start state
 - $q_{\text{accept}} \in Q$ is the accept state
 - $q_{\text{reject}} \in Q$ is the reject state (and $q_{\text{accept}} \neq q_{\text{reject}}$)
 - $\delta: \underbrace{(Q-H) \times \Gamma}_{\uparrow} \rightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function

The tape

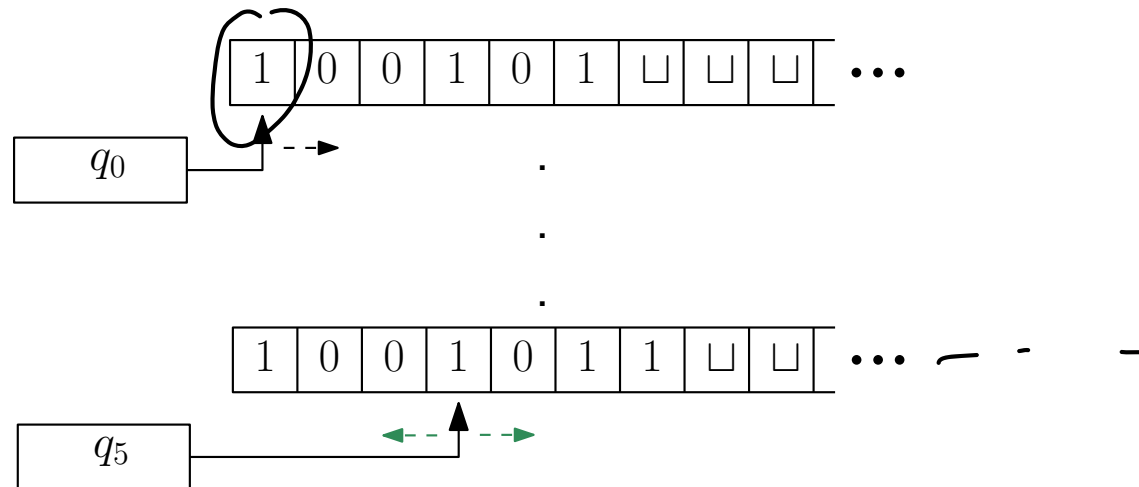
- The tape starts from a left hand side; head at the first cell.
- Starting from the first cell, the tape initially contains the input string in consecutive cells. The rest of the cells contain the blank symbol $\sqcup$. So, initially, the first blank $\sqcup$ marks the end of the input.



- To the right the tape is unbounded, that is, if ever a new tape cell is needed it is always available. But only finitely many are ever non-blank.
- If the tape head is at the first tape cell and the instruction is to move left, the tape head remains where it is, in the first cell.

The tape

- The tape starts from a left hand side; head at the first cell.
- Starting from the first cell, the tape initially contains the input string in consecutive cells. The rest of the cells contain the blank symbol $\square$. So, initially, the first blank $\square$ marks the end of the input.



- To the right the tape is unbounded, that is, if ever a new tape cell is needed it is always available. But only finitely many are ever non-blank.
- If the tape head is at the first tape cell and the instruction is to move left, the tape head remains where it is, in the first cell.

Halt states

- The accept/reject states are called **halt** states.
- Note the transition function explicitly excludes transition from halt states.
- Therefore if a halt state is reached no more transitions are possible.
- Also notice that δ is a function, hence Turing Machines as defined above are *deterministic*.
- Sometimes a Turing Machine will be denoted by TM.

Halt states (cont.)

- The accept/reject states are called **halt** states.
- Note the transition function explicitly excludes transition from halt states.
- Therefore if a halt state is reached no more transitions are possible.
- Also notice that δ is a function, hence Turing Machines as defined above are deterministic.
- Sometimes a Turing Machine will be denoted by TM.

Operation

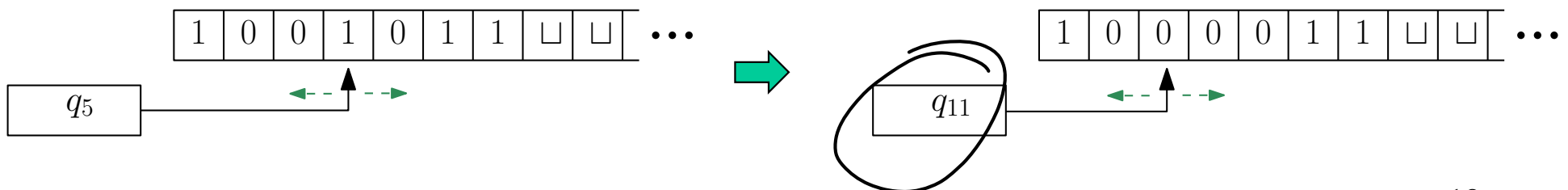
control

- The state part of the TM begins in the initial state.
The tape head is initially placed at the first cell of the tape.
- The TM then moves dependent on the state it is in and the symbol it reads off the tape.
- It will perform a state transition (possibly to the same state) and
 - replace the symbol read with a new symbol (option of not changing the symbol at all)
 - move the tape head one cell to the left or to the right
- E.g, for $\delta(q_5, 1) = (q_{11}, 0, L)$



Operation (cont.)

- The state part of the TM begins in the initial state.
The tape head is initially placed at the first cell of the tape.
- The TM then moves dependent on the state it is in and the symbol it reads off the tape.
- It will perform a state transition (possibly to the same state) and
 - replace the symbol read with a new symbol (option of not changing the symbol at all)
 - move the tape head one cell to the left or to the right
- E.g, for $\delta(q_5, 1) = (q_{11}, 0, L)$



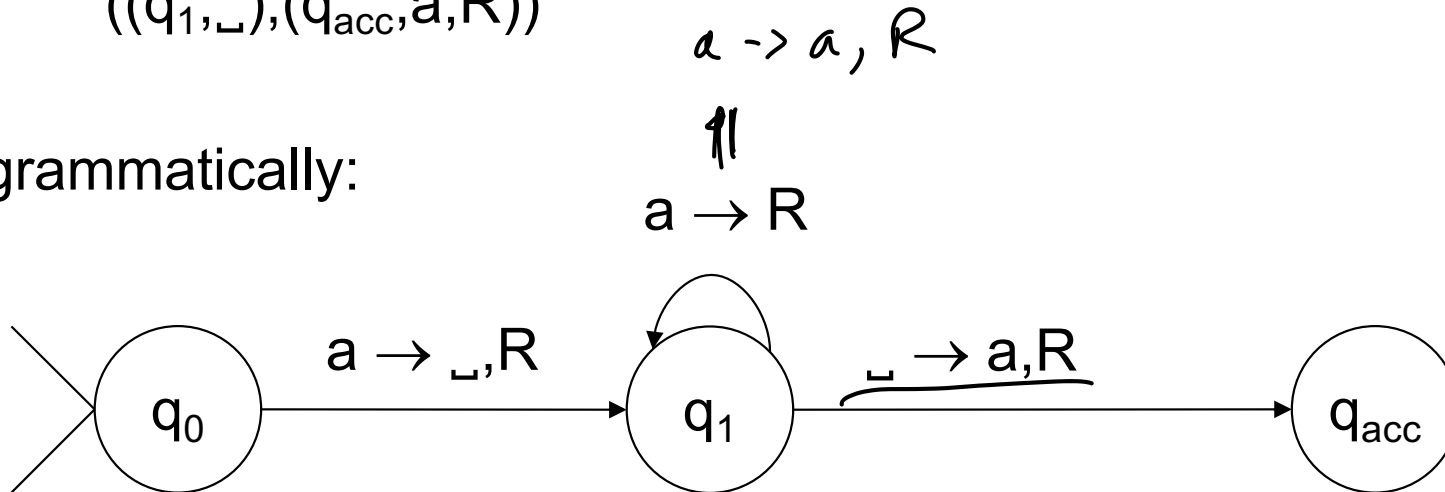
Operation (cont.)

- Finally the TM may reach a halt state and operation ceases.
(if it does not reach a halt state, it goes on forever!)
- Note 1: you can use an underscore '_' instead of '␣'.
- Note 2: the input is written in the cells at the beginning of the tape.

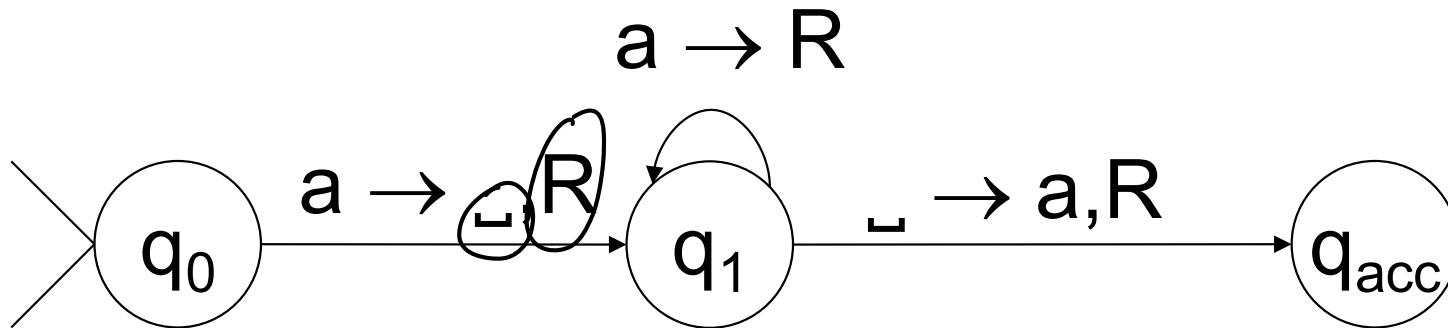
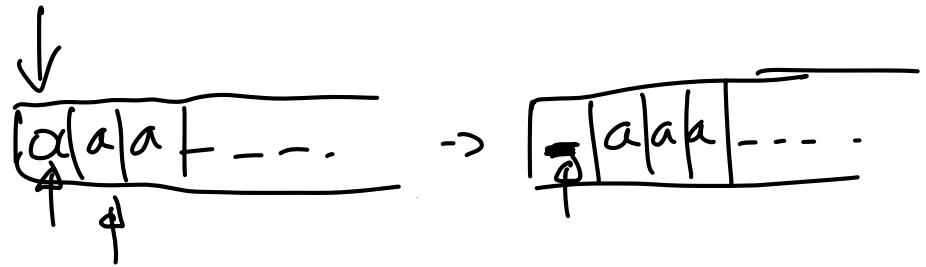
Example

- Consider TM with states $\{q_0, q_1, q_{\text{accept}}\}$, where q_0 is the initial state (and q_{reject} is not reachable) and $\Sigma = \{a\}$, $\Gamma = \{a, \sqcup\}$.
- The transition function consists of just three transitions:
 - $((q_0, a), (q_1, \sqcup, R))$
 - $((q_1, a), (q_1, a, R))$
 - $((q_1, \sqcup), (q_{\text{acc}}, a, R))$

Diagrammatically:



Example (cont.)



- This TM has the effect of moving a string of 'a's one cell to the right.
- (Note: if the cell doesn't change, omit the write part, as in: $a \rightarrow R$)

Computation

- A **configuration** is formally a triple consisting of the current state, the cell currently being observed by the head, and the current tape contents.

E.g., $(q, _a\textcolor{red}{a}_, _aaa_)$

- This is often written as the contents of the tape, with the state symbol before the current cell

E.g., $_a\textcolor{red}{a}_$

$q _ _ a a a _ _$

- If the state is a halt state, then this will be described as a **halting** configuration.

Example

- Consider the operation of the machine:
 - $q_0 a a _$ (read first cell, replace by blank, move right)
 - $_ q_1 a a _$ (read second cell, move right)
 - $_ a q_1 a _$ (read third cell, move right)
 - $_ a a q_1 _$ (read fourth cell, replace by 'a' move right)
 - $_ a a a q_{\text{accept}} _$ (halt)

Example

- Consider the operation of the machine:
 - $q_0 a a _$ (read first cell, replace by blank, move right)
 - $_ q_1 a a _$ (read second cell, move right)
 - $_ a q_1 a _$ (read third cell, move right)
 - $_ a a q_1 _$ (read fourth cell, replace by 'a' move right)
 - $_ a a a q_{\text{accept}} _$ (halt)

Example

- Consider the operation of the machine:
 - $q_0 \text{a} \text{a} _$ (read first cell, replace by blank, move right)
 - $_ q_1 \text{a} _$ (read second cell, move right)
 - $_ \text{a} q_1 \text{a} _$ (read third cell, move right)
 - $_ \text{a} _ q_1 _$ (read fourth cell, replace by 'a' move right)
 - $_ \text{a} \text{a} _ q_{\text{accept}} _$ (halt)

Example

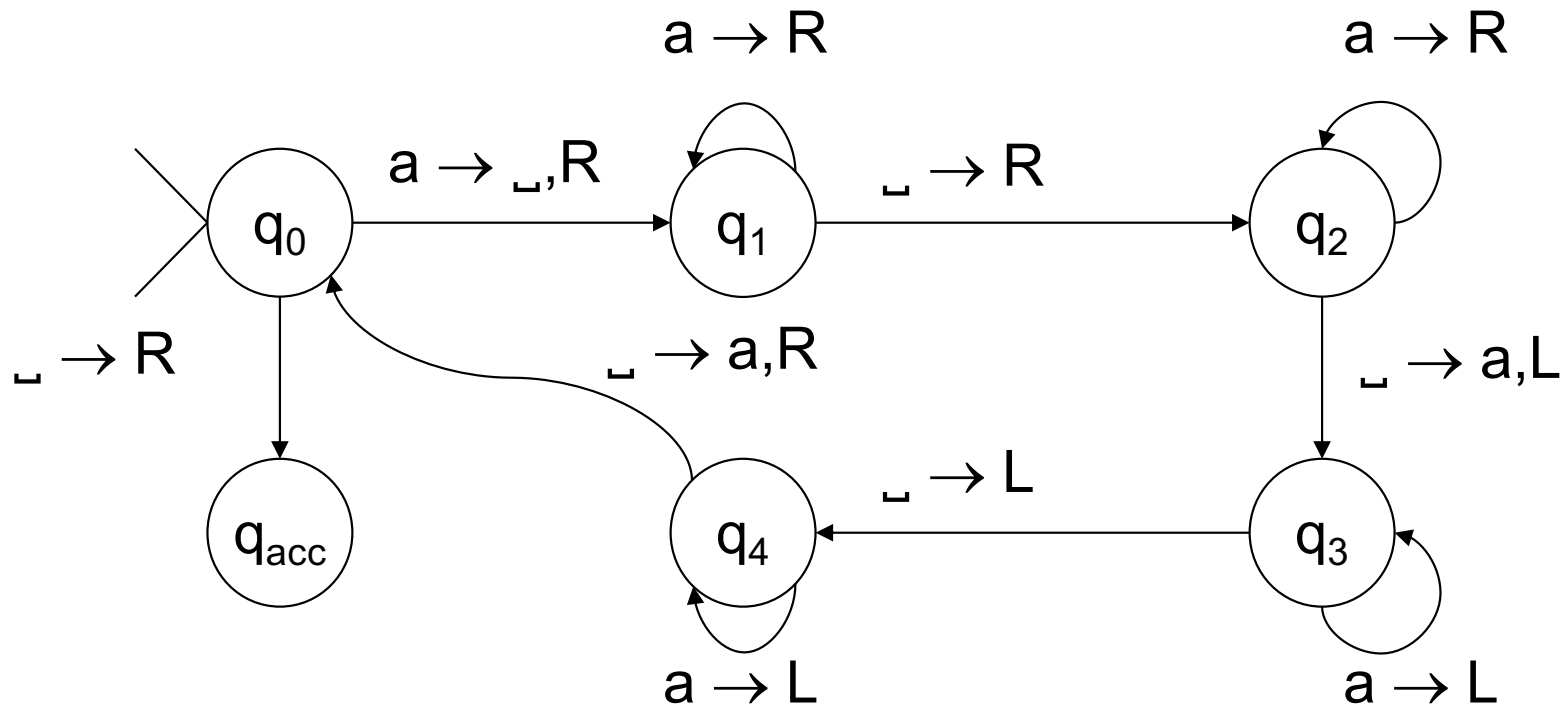
- Consider the operation of the machine:
 - $q_0 a a _$ (read first cell, replace by blank, move right)
 - $_ q_1 a a _$ (read second cell, move right)
 - $_ a q_1 a _$ (read third cell, move right)
 - $_ a a q_1 _$ (read fourth cell, replace by 'a', move right)
 - $_ a a a q_{\text{accept}} _$ (halt)

Computation (cont.)

- Suppose one configuration C_1 leads to configuration C_2 by application of a single transition, this will be denoted $C_1 \vdash C_2$.
- A chain of these will be denoted $C_1 \vdash^* C_2$.
- Such a sequence is called a **computation** by the TM, and the number of transitions is its **length**.

Example

- Consider the following machine that will copy a string one blank space beyond the end of the original copy:



Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0 \underline{a} a _ _$

$_ q_1 a _$

$_ a q_1 _$

$_ a _ q_2 _$

$_ a q_3 a$

$_ q_4 a _ a$

$q_4 _ a _ a$

$a q_0 a _ a$

$a _ q_1 _ a$

$a _ _ q_2 a$

$a _ _ a q_2 _$

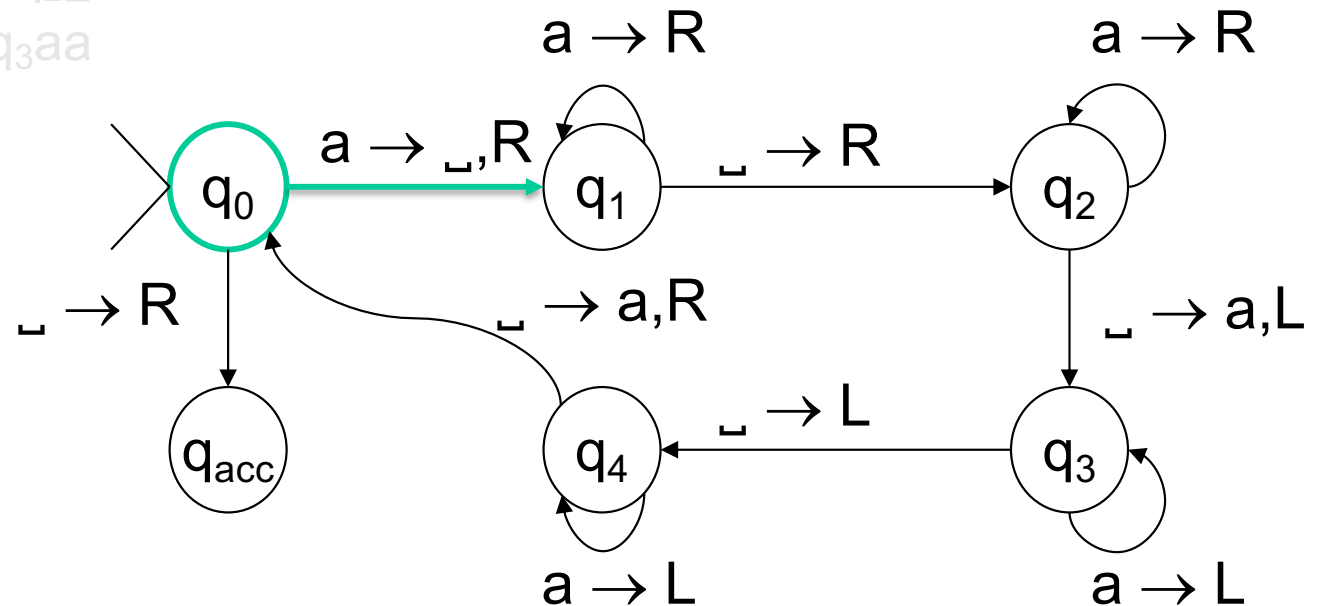
$a _ _ q_3 a a$

$a _ q_3 _ a a$

$a q_4 _ _ a a$

$a a q_0 _ a a$

$a a _ q_{\text{accept}} a a$



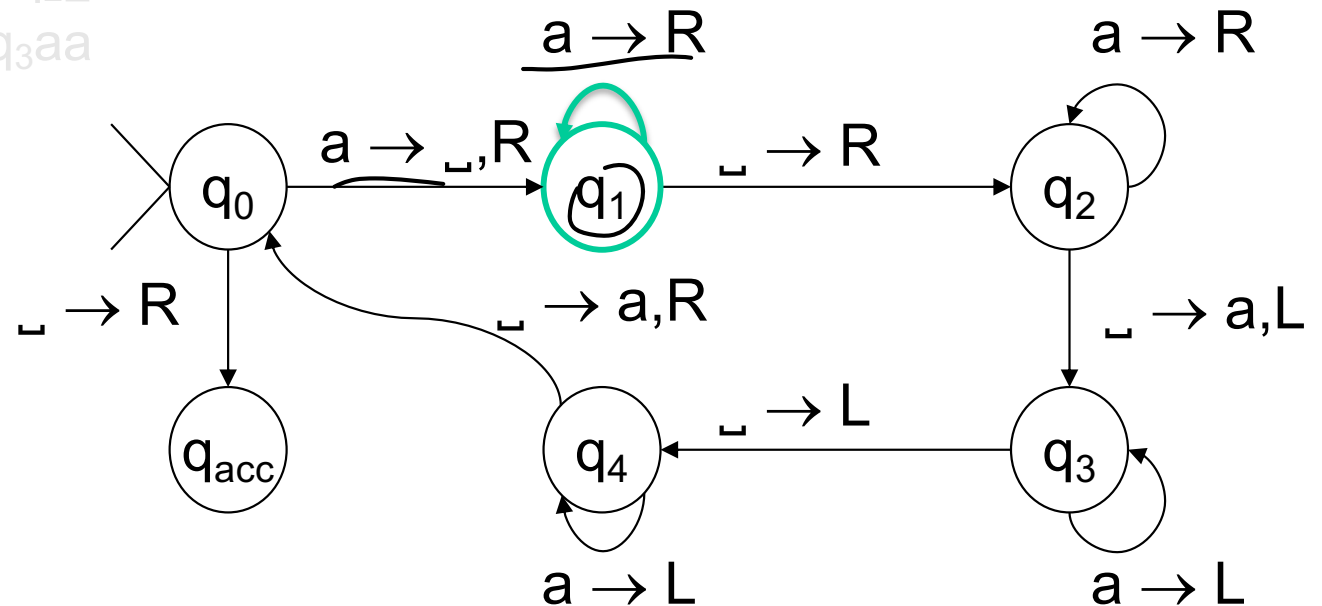
Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$
 $_q_1a_$
 $_a q_1_$
 $_a_ q_2_$
 $_a q_3_a$
 $_ q_4a_a$

$q_4_a_a$
 $a q_0a_a$
 $a_ q_1_a$
 $a_ q_2a$
 $a_a q_2_$
 $a_ q_3aa$

$a_ q_3_aa$
 $a q_4_aa$
 $aa q_0_aa$
 $aa_ q_{accept}aa$



Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$

$_ q_1a_$

$_a \underline{q_1}_$

$_a_ q_2_$

$_a q_3_a$

$_ q_4a_a$

$q_4_a_a$

$a q_0a_a$

$a_ q_1_a$

$a_ q_2a$

$a_a q_2_$

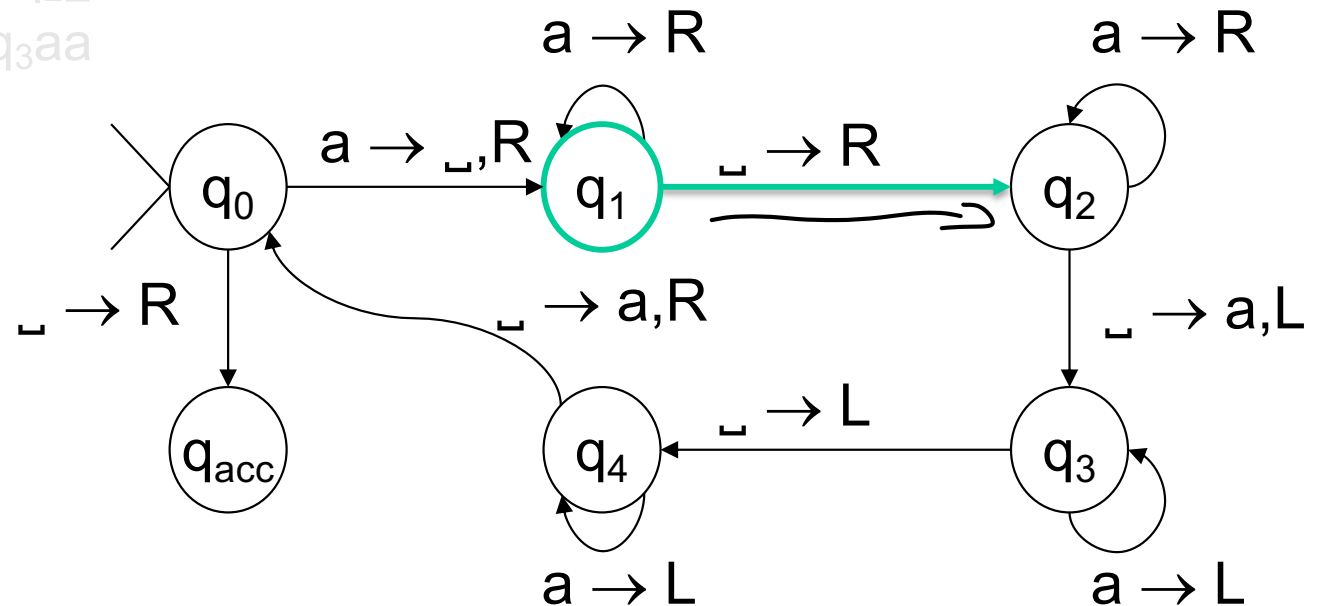
$a_ q_3aa$

$a_ q_3_aa$

$a q_4_aa$

$aa q_0_aa$

$aa_ q_{accept}aa$



Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0 \underline{a} a _ _$

$_ q_1 a _$

$_ a q_1 _$

$_ a _ q_2 _$

$_ a q_3 _ a$

$_ q_4 a _ a$

$q_4 _ a _ a$

$a q_0 a _ a$

$a _ q_1 _ a$

$a _ _ q_2 a$

$a _ _ a q_2 _$

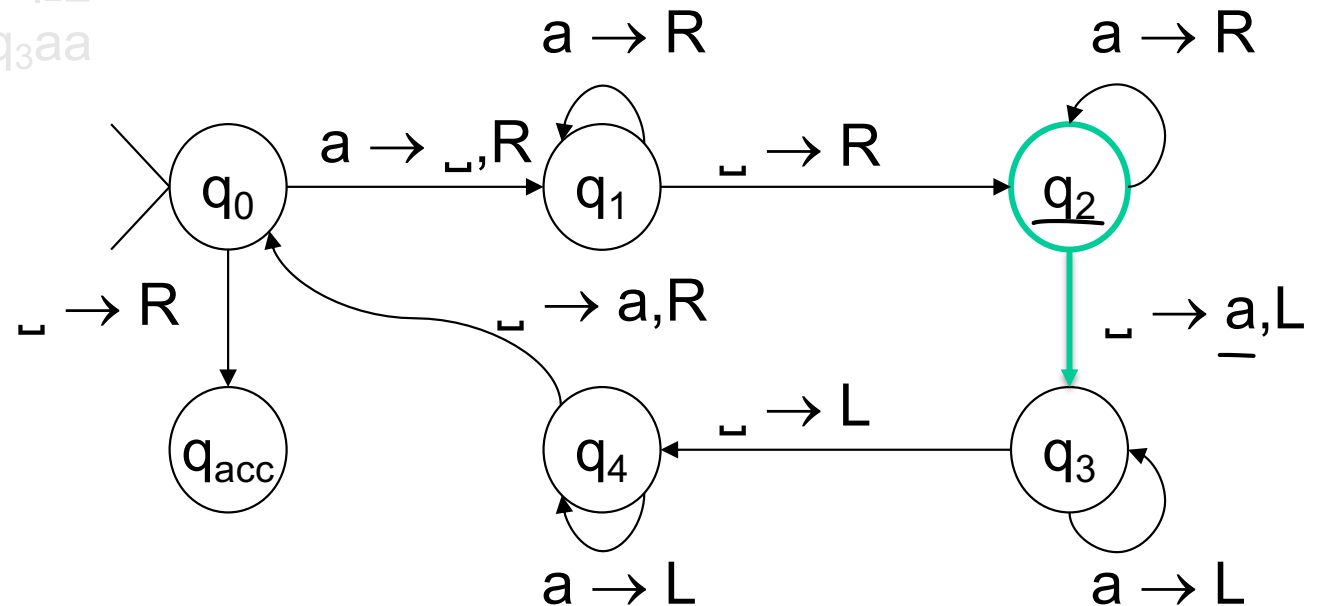
$a _ _ q_3 a a$

$a _ q_3 _ a a$

$a q_4 _ _ a a$

$a a q_0 _ a a$

$a a _ q_{\text{accept}} a a$



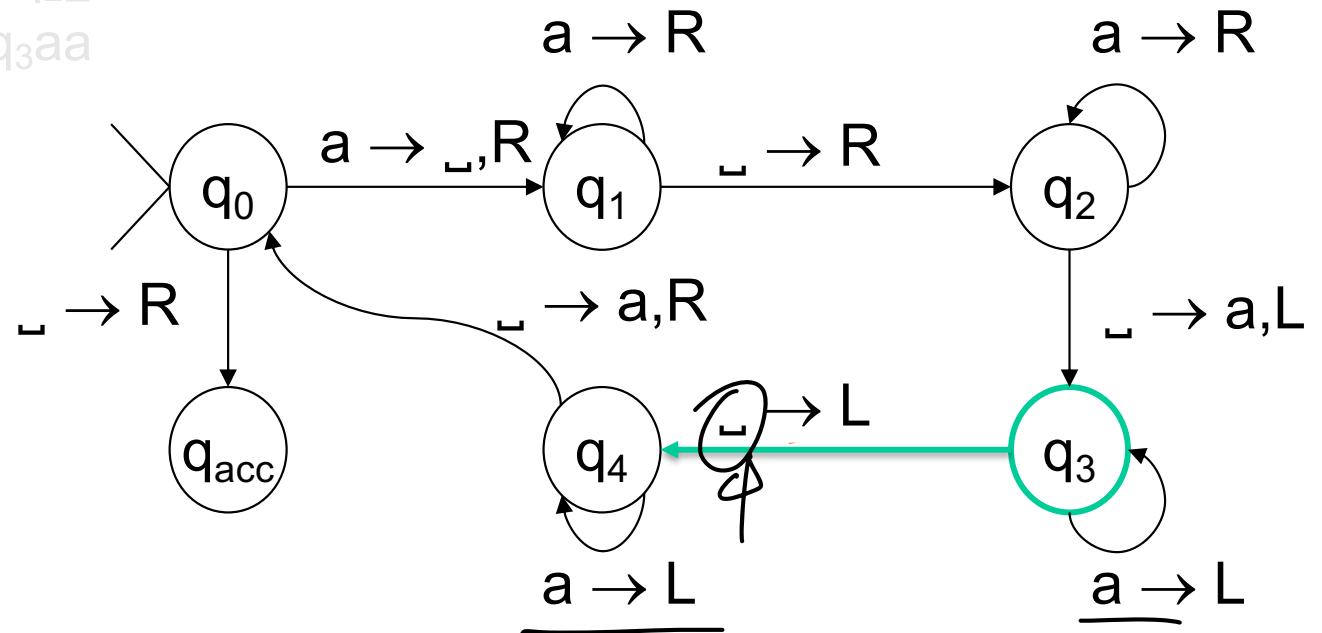
Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$
 $_ q_1a_$
 $_a q_1_$
 $_a_ q_2_$
 $_a q_3_a$
 $_ q_4a_a$

$q_4_a_a$
 $a q_0a_a$
 $a_ q_1_a$
 $a_ q_2a$
 $a_a q_2_$
 $a_ q_3aa$

$a_ q_3_aa$
 $a q_4_aa$
 $aa q_0_aa$
 $aa_ q_{accept}aa$



Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$

$_ q_1a_$

$_a q_1_$

$_a_ q_2_$

$_a q_3_a$

$_ q_4a_a$



$q_4_a_a$

$a q_0a_a$

$a_ q_1_a$

$a_ a q_2a$

$a_ a q_2_$

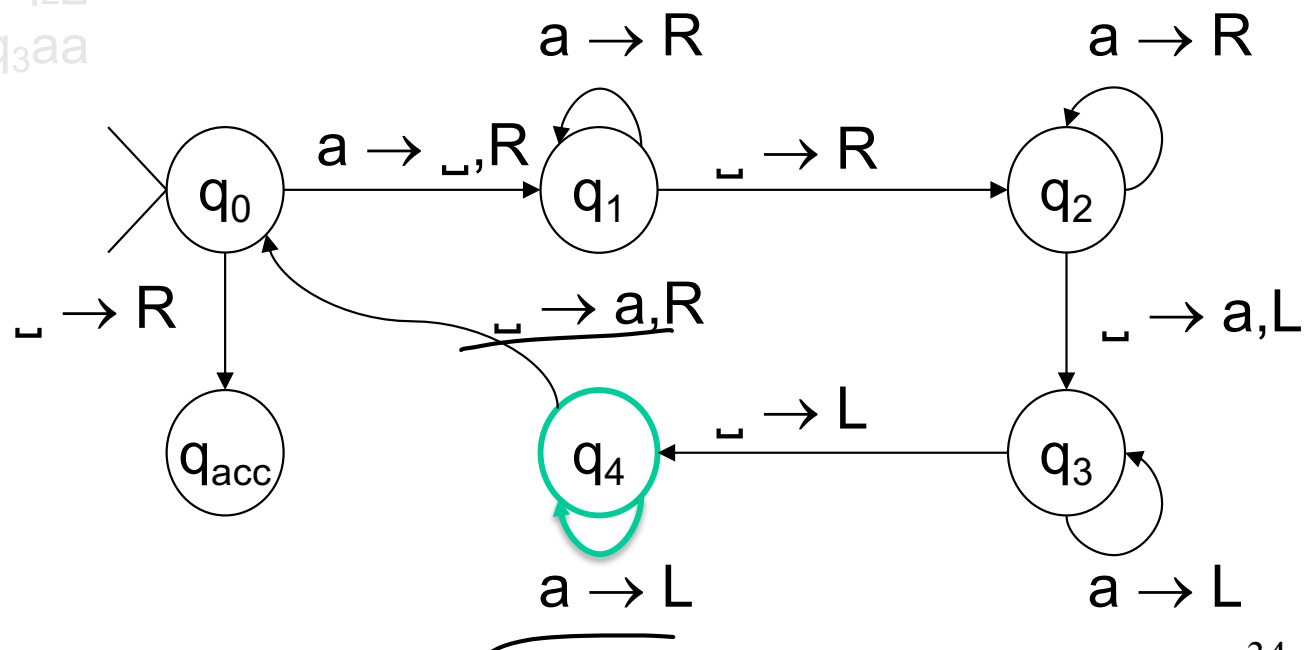
$a_ a q_3aa$

$a_ q_3_aa$

$a q_4_ a$

$aa q_0_aa$

$aa_ q_{accept}aa$



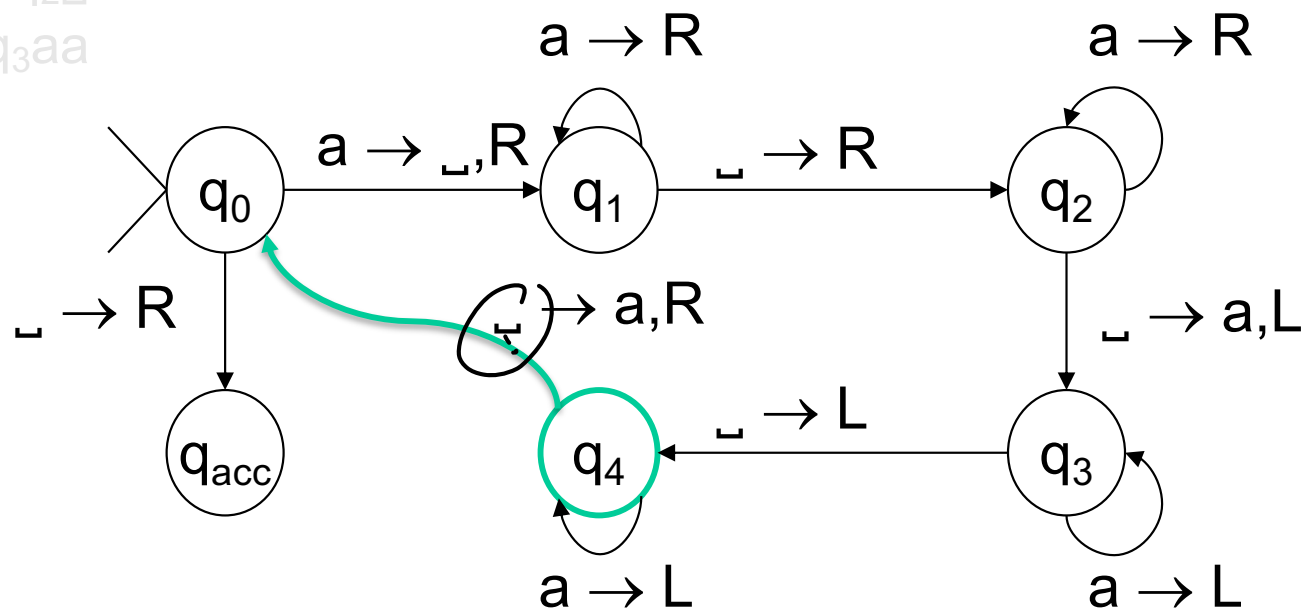
Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$
 $_ q_1a_$
 $_a q_1_$
 $_a_ q_2_$
 $_a q_3_a$
 $_ q_4a_a$

$\downarrow$
 $q_4_a_a$
 $a q_0a_a$
 $a_ q_1_a$
 $a_ q_2a$
 $a_a q_2_$
 $a_ q_3aa$

$a_ q_3_aa$
 $a q_4_aa$
 $aa q_0_aa$
 $aa_ q_{accept}aa$



Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$

$_ q_1a_$

$_a q_1_$

$_a_ q_2_$

$_a q_3_a$

$_ q_4a_a$

↑

and so on..

$q_4_a_a$

$a \underline{q_0}a_a$

$a_ q_1_a$

$a_ _ q_2a$

$a_ _a q_2_$

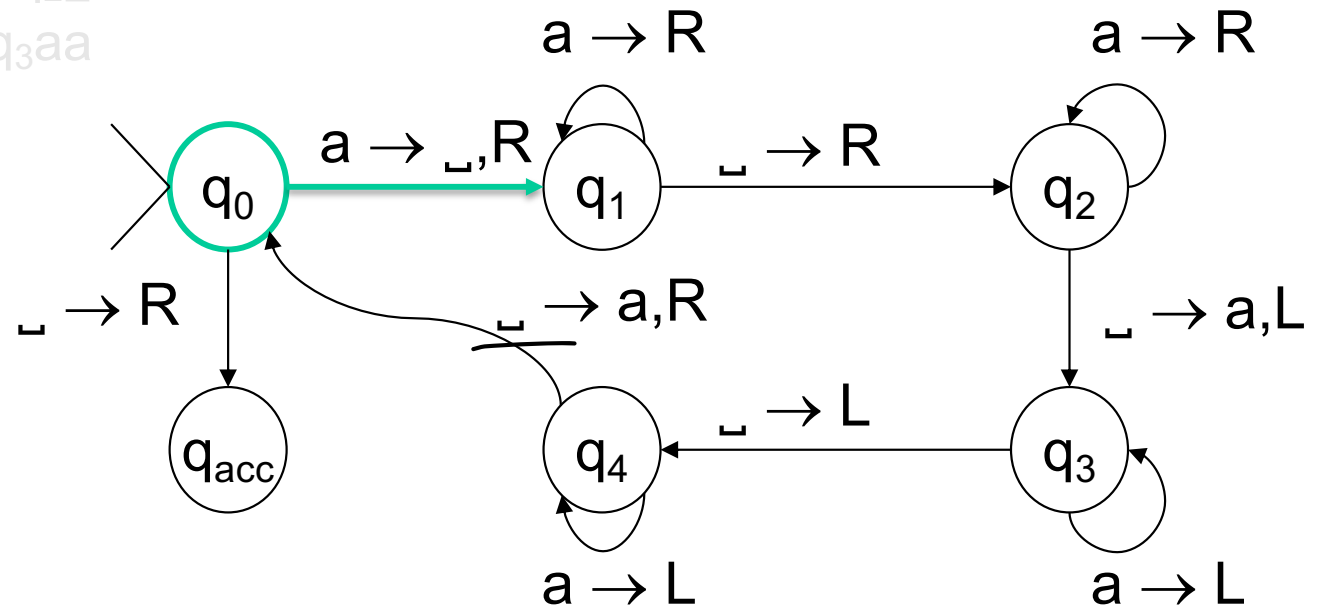
$a_ _ q_3aa$

$a_ q_3_aa$

$a _ q_4_ _aa$

$aa _ q_0_aa$

$aa_ q_{accept}aa$



Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$

$_ q_1a_$

$_a q_1_$

$_a_ q_2_$

$_a q_3_a$

$_ q_4a_a$

$q_4_a_a$

$a q_0a_a$

$a_ q_1_a$

$a_ q_2a$

$a_a q_2_$

$a_ q_3aa$

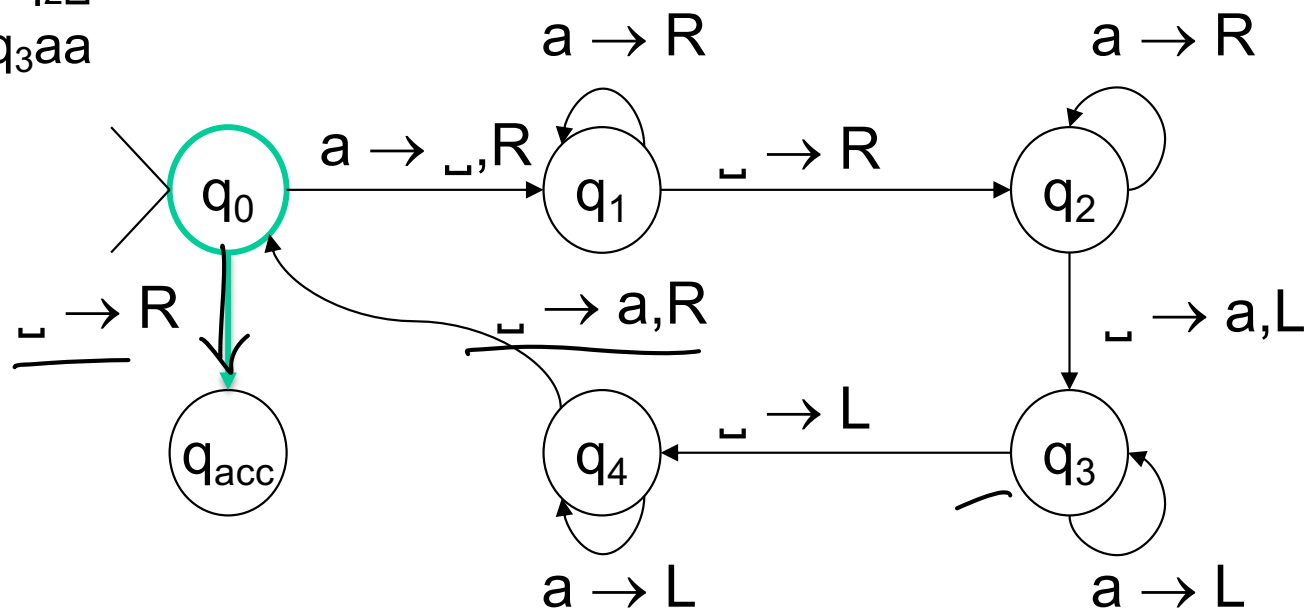
$a_ q_3_aa$

$a q_4_a$

$aa q_0_aa$

$aa_ q_{accept}aa$

...until



Example (cont.)

- For the input being a string of two 'a's this gives the following computation:

$q_0aa_$

$_ q_1a_$

$_a q_1_$

$_a_ q_2_$

$_a q_3_a$

$_ q_4a_a$

$q_4_a_a$

$a q_0a_a$

$a_ q_1_a$

$a_ q_2a$

$a_a q_2_$

$a_ q_3aa$

$a_ q_3_aa$

$a q_4_aa$

$aa q_0_aa$

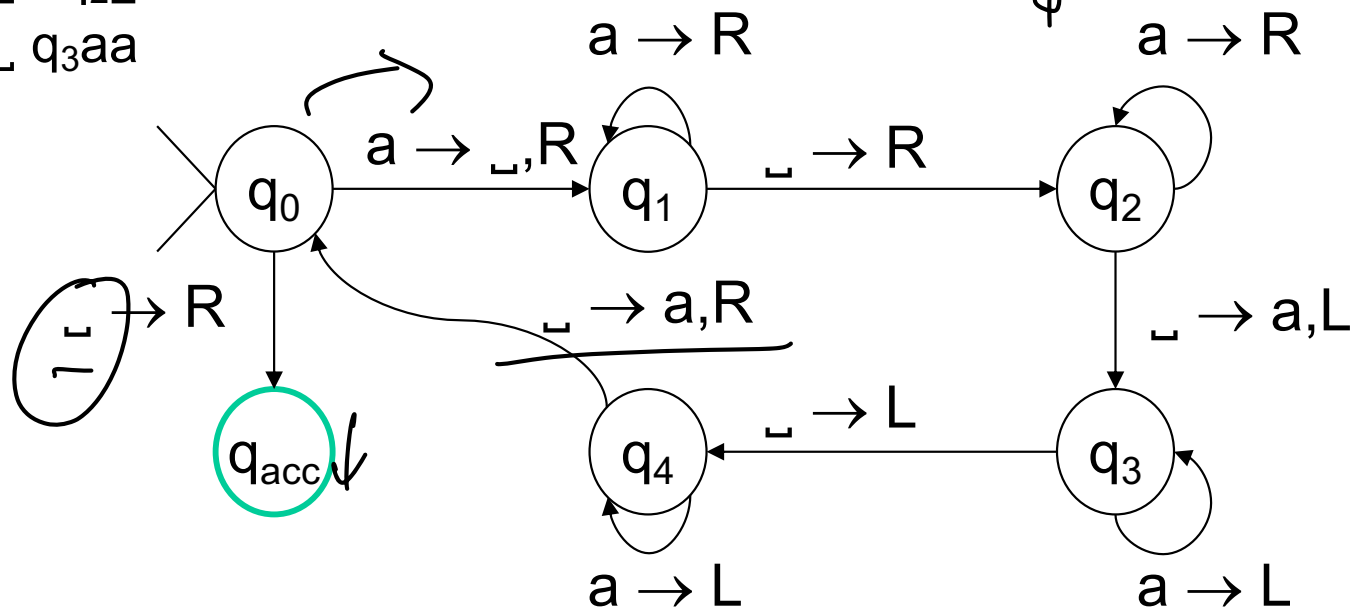
$aa_ q_{\text{accept}}aa$

$aa_$

$_ : _a$

a_aa_a

a_a_a



Output: definition

- Let M be a TM. Let $\underline{w}$ be a string over $\underline{\Sigma}$.
- Suppose that starting from initial configuration $q_0 \underline{w}$, M halts in configuration $\underline{u} h v$, for some halting state h , where $\underline{u}, \underline{v}$ are strings over Γ .
 ~~q_0~~ ~~q_1~~ ~~q_2~~
- Then $\underline{uv}$ is called the **output** of machine M on the input $\underline{w}$.
denoted $M(w)$.

E.g., in the example before, $\underline{q_0 aa} \vdash^* aa_ q_{\text{accept}} aa$, so

$M(w) = aa_aa$, where $w = aa$

$.aa_aa---$

Turing-recognisable Languages

- **Definition:** The collection of strings accepted by TM M is called the language **recognised** by M or $L(M)$.
- **Definition:** A language is called **Turing-recognisable*** if there exists a TM that recognises it.
- *or semi-decidable or Recursively Enumerable

Turing-decidable Languages

- **Definition:** A TM that halts on all inputs is called a **decider**, and a decider that recognises some language is said to decide it.
- **Definition:** A language is called **Turing-decidable*** if some TM decides it.
- That is, a TM for a Turing-decidable language halts for all inputs.

* or Turing Computable or Recursive

Example 0

$\Sigma = \{0, 1\}$, $L = \{\underline{0^n 1^n} \mid \underline{n \geq 0}\}$ Design a TM for L

- Check a few cases

0011 $\in L$ (should lead to q_{acc})

000111 $\in L$ -||-

empty string $\rightarrow \epsilon \in L$ -||-

010 $\notin L$ (should reject)

1100 $\notin L$ -||-

Example 0

$\Sigma = \{0, 1\}$, $L = \{0^n 1^n \mid n \geq 0\}$ Design a TM for L

- Recursive approach

(i) ϵ (empty) $\in L$

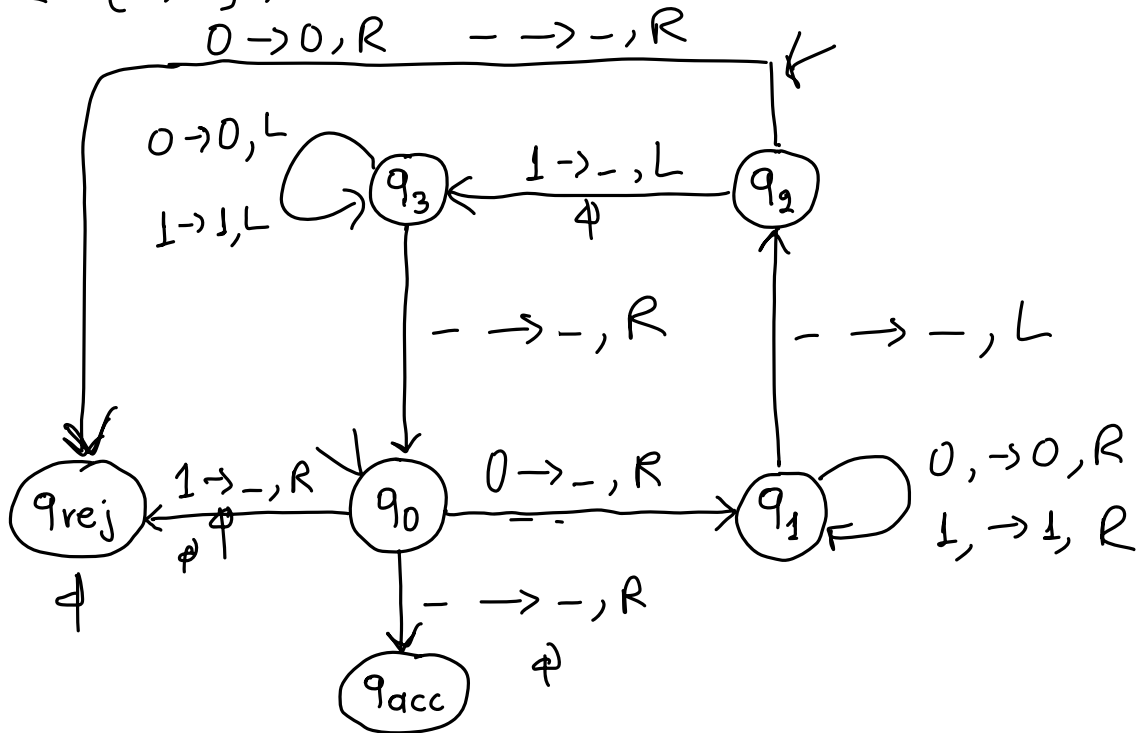
(ii) $0w1 \in L \iff ? \quad w \in L$

(iii) any string w starting with 1 is not in L

(iv) any string w ending with 0 is not in L

Example 0

$\Sigma = \{0, 1\}$, $L = \{0^n 1^n \mid n \geq 0\}$ Design a TM for L



0011^d
 = 011_↑1_↑

↓ ↑
 00111:
 — (01) —
 ↑
 ;
 q_{acc}

00001111
 ↑ ↑ ↗ ↘

Example 1

Consider the language $L = \{\underline{w\#w} \mid w \in \{0, 1\}^*\}$.

01011 # 01011
- 1 1 - 1

High-level description (*algorithm*) of a TM deciding L:

- Zig-zag across the tape to corresponding positions on either side of # to check whether these positions contain the same symbol.
- (1) If they don't, or if no # is found, **reject**.
Cross off symbols as they are checked to keep track of which symbols correspond.
- When all symbols to the left of # have been crossed off, check for any remaining symbols to the right of #.
 If any symbols remain, **reject**; otherwise **accept**.

Example 1 (cont.)

Consider the language $L = \{w\#w \mid w \in \{0, 1\}^*\}$.

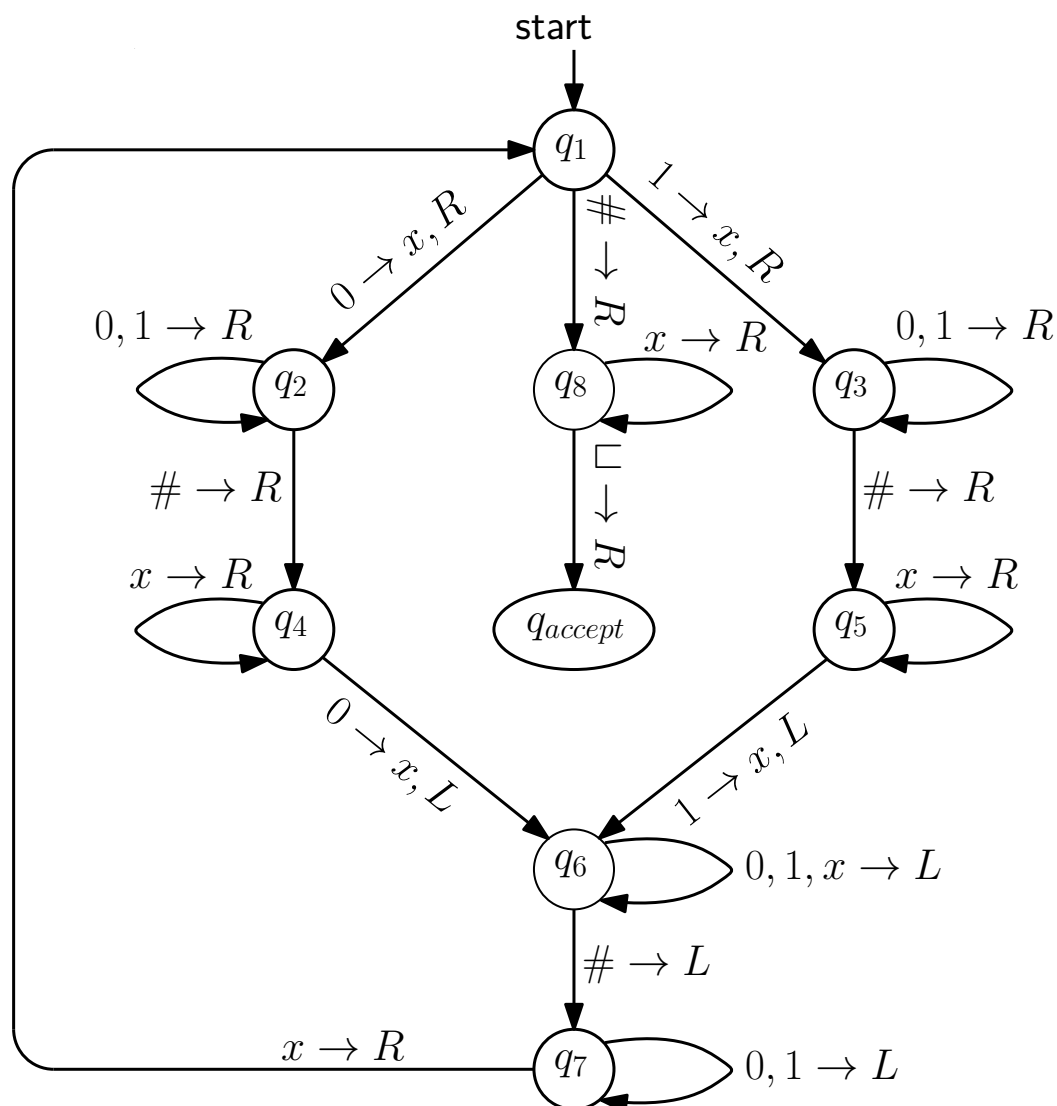
$w \# w'$
 $w' \neq w$

High-level description (*algorithm*) of a TM deciding L :

- Zig-zag across the tape to corresponding positions on either side of $\#$ to check whether these positions contain the same symbol.
- (1) If they don't, or if no $\#$ is found, **reject**.
Cross off symbols as they are checked to keep track of which symbols correspond.
- When all symbols to the left of $\#$ have been crossed off, check for any remaining symbols to the right of $\#$.
- (2) If any symbols remain, **reject**; otherwise **accept**.



Example 1 (cont.)

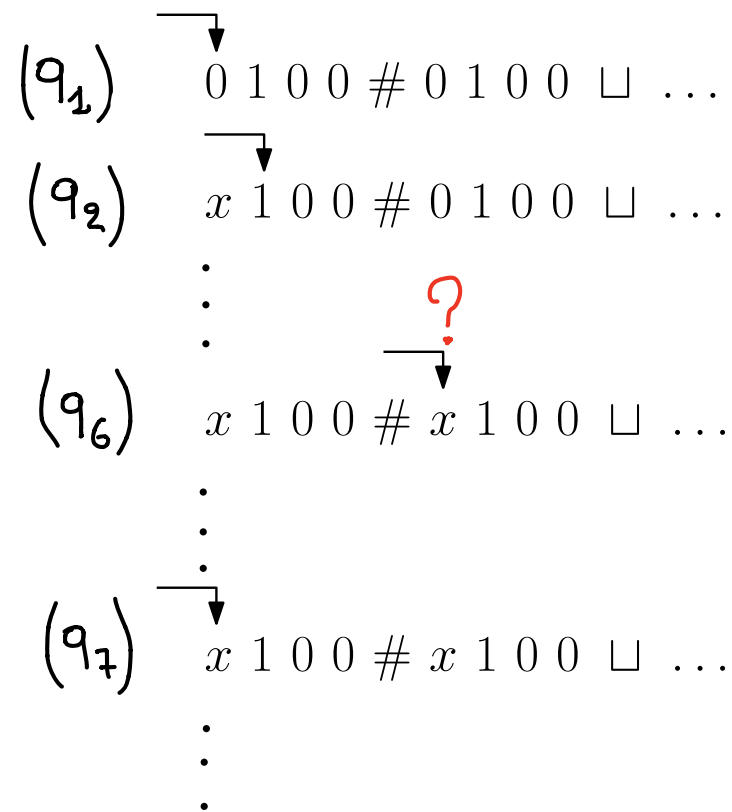
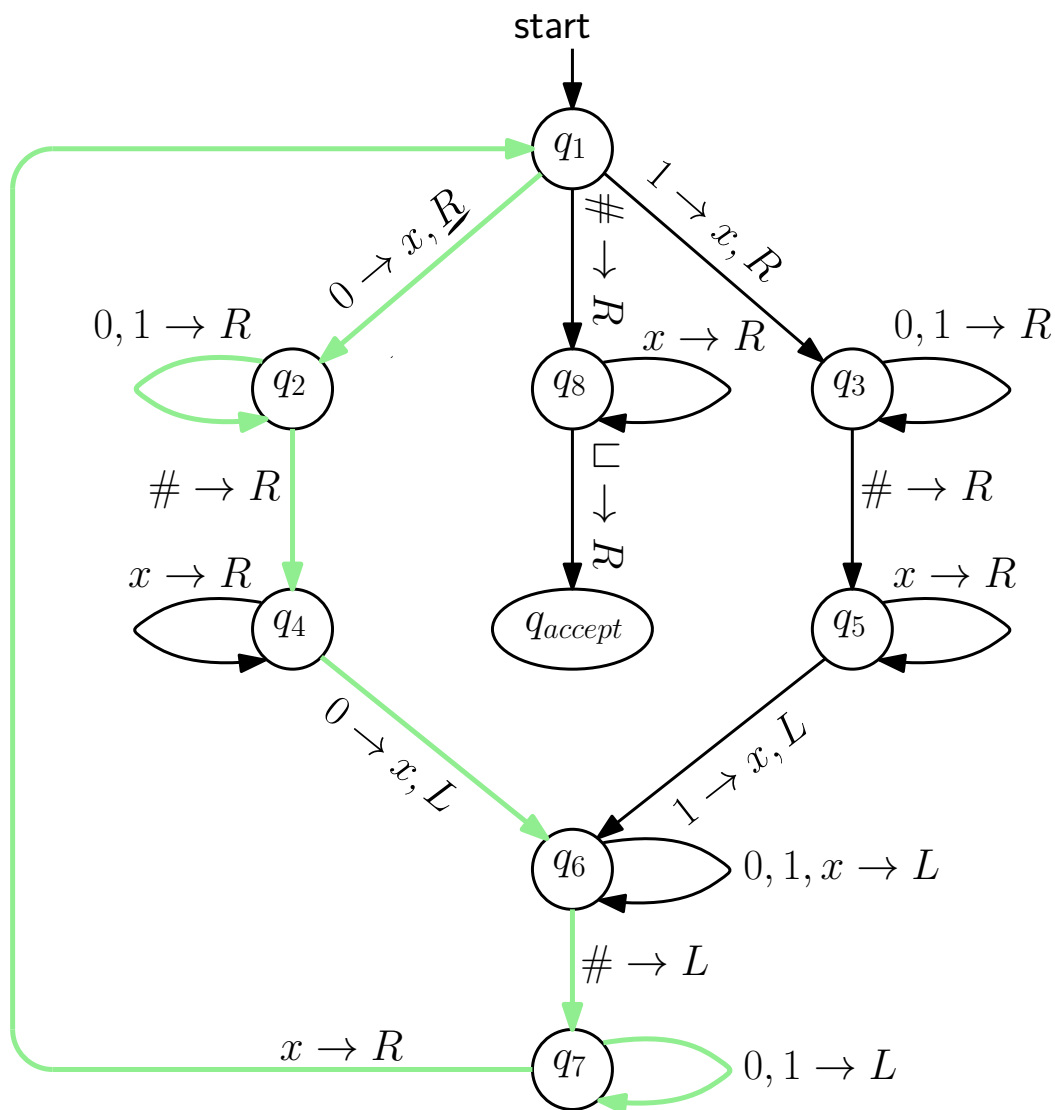


Note : q_{reject} is not shown

- Transitions to q_{reject} are the ones missing for each state and each symbol.
- E.g. the following transitions are also 'present' :
 - q_2 'reads' $\sqcup \rightarrow q_{\text{reject}}$
 - q_4 'reads' 1 or $\sqcup$ or $\# \rightarrow q_{\text{reject}}$
 - q_8 'reads' $0, 1$ or $\# \rightarrow q_{\text{reject}}$ etc.

Example 1 (cont.)

Example run for input: 0100#0100



stage (1): $q_1 - q_7$
stage (2): q_1, q_7, q_8 } Algorithm (slide 43)

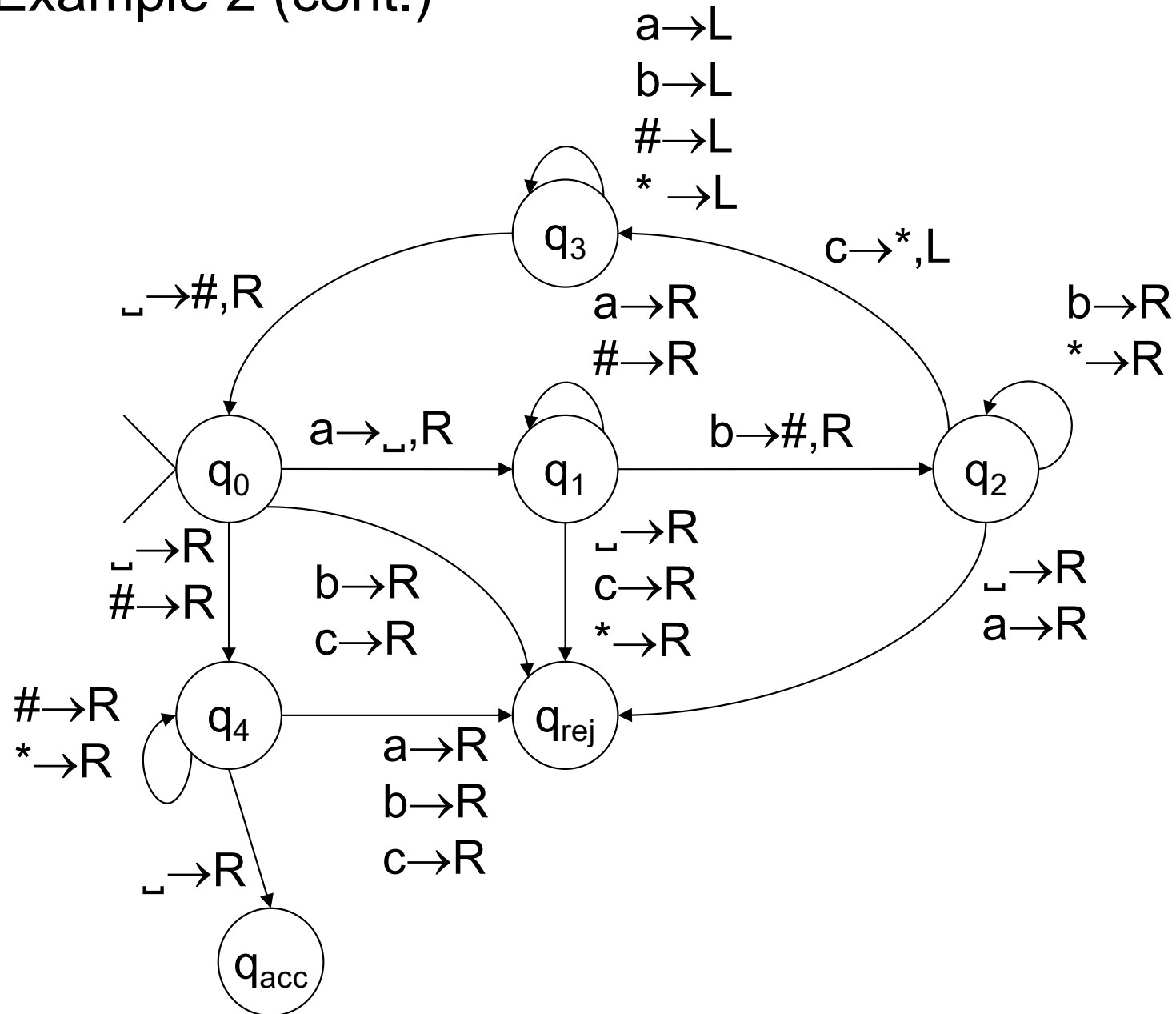
Example 2

not context-free



- Consider the language $L = \{a^n b^n c^n \mid n \geq 0\}$.
- Input tape will be of form w , where $w \in \{a, b, c\}^*$.
- A machine operating as on the following slide will recognise L .
- (Note that the machine has a larger alphabet Γ than L 's input alphabet Σ . It is assumed that these symbols are not initially written on the tape.)

Example 2 (cont.)



Computing a function

- Suppose f is a function from Σ to Σ , with domain $\text{Dom}(f)$.
- It is said that M **computes** function f if M accepts for every $w \in \text{Dom}(f)$ with $M(w)=f(w)$ and M rejects (or does not halt) for $w \notin \text{Dom}(f)$.

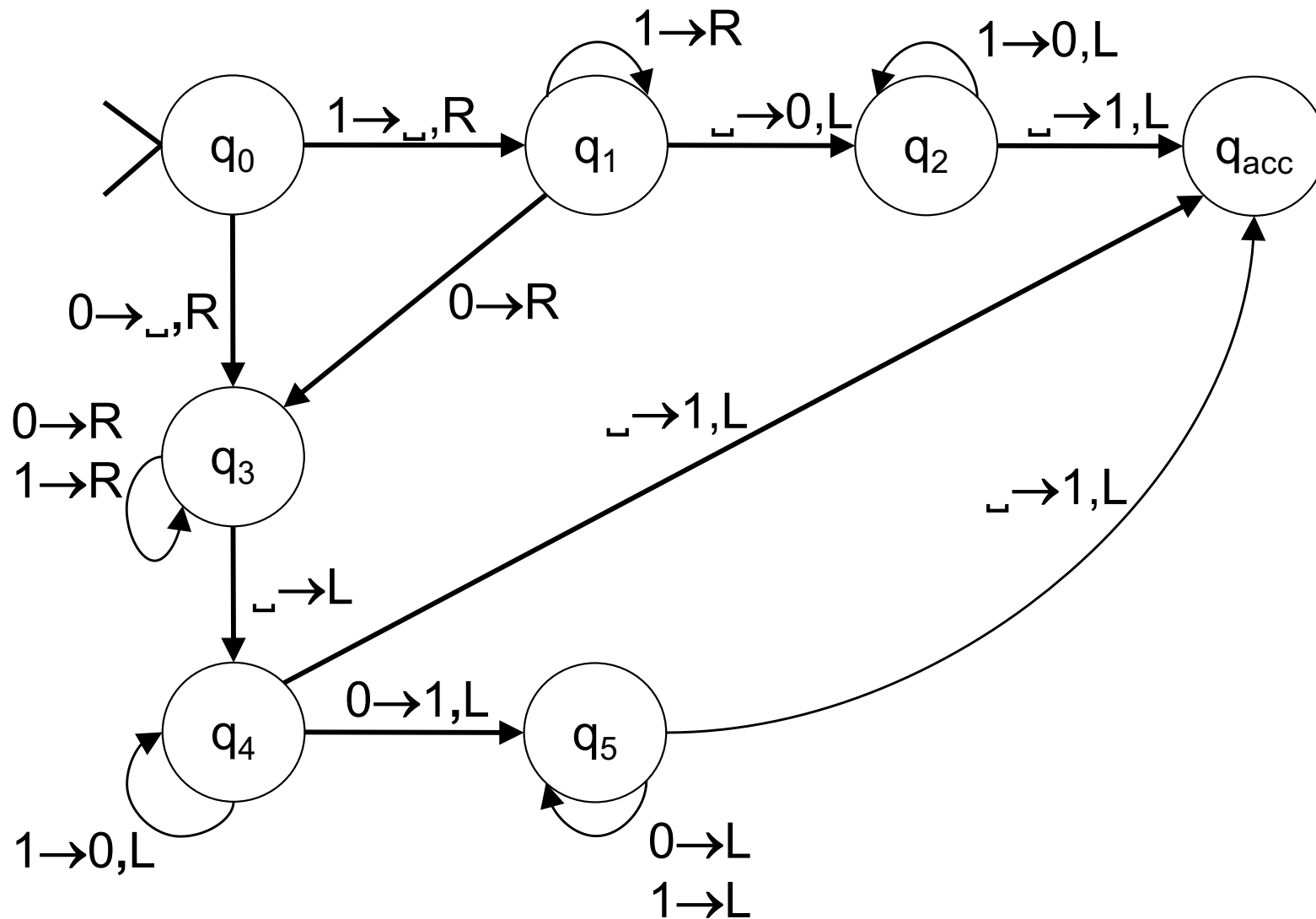
Numeric functions

- Words consisting of strings of '0's and '1's can be considered at binary numbers.
- A TM can work with these numbers and compute with them, leaving an output on the tape.
- This ability to compute functions gives TMs much more power than other classes of machine covered in this course !

Example

- Consider the simple example of the function $f(k) = k+1$.
- A TM to calculate this is outlined (algorithmically) below:
 - Move to the right end of the input string.
 - Moving left, it flips '1's to '0's until it reaches its first zero, when it writes '1' and halts.
 - The only case not covered is if all cells are '1'. Then write '1' for the first (on the left) cell and add an additional zero in the first blank on the right before halting.

Example (cont.)



Summary

- Turing machines are a powerful formal machine that can not only recognise languages, but compute outputs (and functions).
- Building them can be approached similarly to programming.

Reading

- Sipser, 139–149.
- Exercises 3.1,3.2,3.5,3.7,3.8.